Examining ASP.NET's Membership, Roles, and Profile - Part 8

By Scott Mitchell

Introduction

One of the main challenges of building a programming framework is balancing the desires to create a standardized, straightforward API for accomplishing common tasks and providing flexibility and customizability so that developers using the framework can mold the framework to fit their applications rather than having to bend their applications to fit the framework. With the .NET Framework 2.0, Microsoft implemented this delicate balance with the *provider model*. The provider model allows for a standardized API to be defined within the framework, but enables developers to design their own concrete implementation and plug that implementation into their systems at runtime. In Part 7 of this article series we saw how the Membership, Roles, and Provider pieces examined throughout this series all utilize the provider model. Moreover, in Part 7 we created a custom XML provider for the Profile system.

I recently worked on a website that primarily contained static content. The client, however, had a particular page that needed to display data from a simple database with just one table. Additionally, a web page was needed to allow a set of administrators to add, update, and delete data from this table. With ASP.NET 2.0's data source controls and Membership system, this functionality is typically a breeze, but there was a catch - the web hosting company didn't support SQL Server databases, so Microsoft Access databases needed to be used instead. The challenge here is that the .NET Framework BCL only contains a Membership provider for *Microsoft SQL Server*.

Fortunately, Microsoft provides an Access database template and providers for using Membership, Roles, and Profile with Access. In this article, we'll look at how to get these Access-based providers and how to use the provider in an ASP.NET 2.0 web application. Read on to learn more!

Getting and Using the Sample Access Providers

In the Beta stages of ASP.NET 2.0, Microsoft included both Access- and SQL Server-based providers for Membership, Roles, and Profile as part of the base class libraries of the .NET Framework. By the time ASP.NET 2.0 was officially released, however, Microsoft decided to not include built-in Access support and instead encouraged hobbyists, novices, and other folks who might choose Access to use Microsoft SQL Server 2005 Express Edition instead. Unfortunately, not everyone can move to SQL Server 2005. Many web hosting companies do not support the Express Edition. People upgrading to 2.0 might already have a mature data-driven application that uses Access, and porting to SQL Server may not be possible due to time constraints, budget, or interoperability with other applications.

Fortunately, Microsoft has made the Access providers from the beta stages available as a separate download. The download is a Microsoft Visual Studio Installer file (VSI) that includes C# class files and an Access database that provides the desired functionality. You can download the Microsoft Sample Access Providers directly from Microsoft here. The VSI file is also available for download at the end of this article.

There are three steps we must tackle in order to start using the Access providers:

1. **Get the Access Provider Classes into the Web Application** - the C# class files in the VSI download need to either be compiled into an assembly and that assembly needs to be copied to the web application's `/bin` folder *or* the class files themselves need to be copied to the

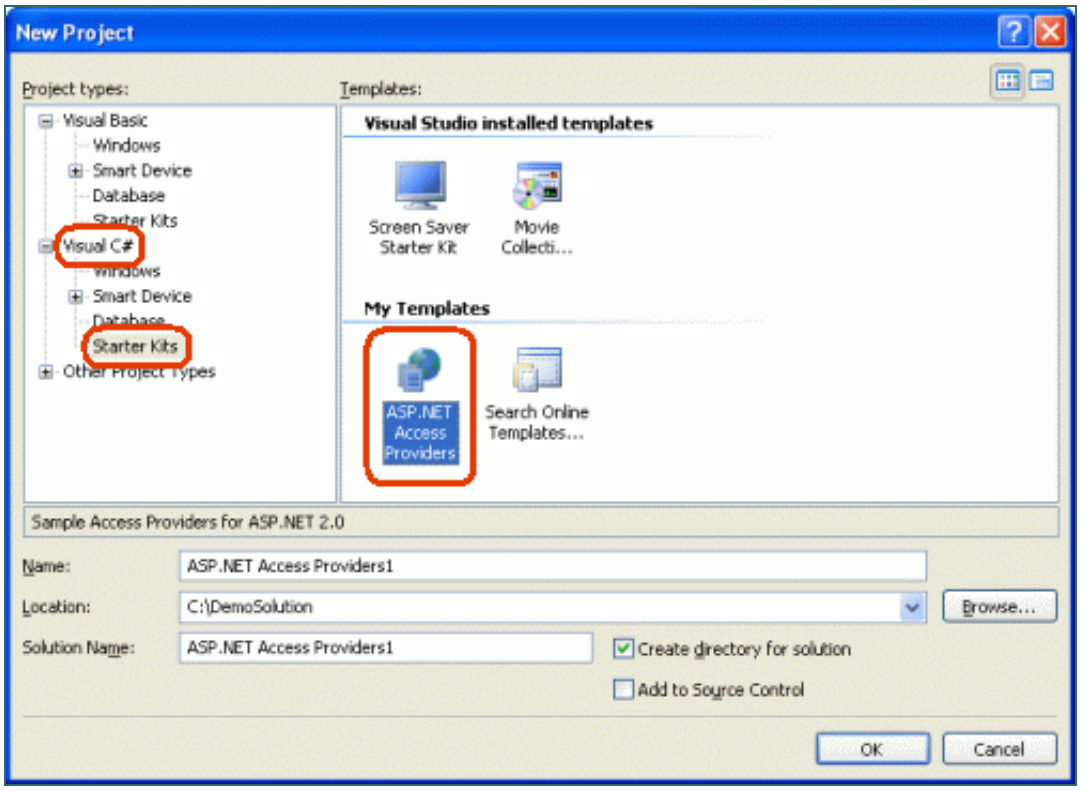
application's `App_Code` folder,

2. **Ready the Database** - we need to make the Access database available to the web application, and
3. **Update `Web.config`** - the `Web.config` must be configured so that the application uses the Access providers for Membership, Roles, and Profile rather than the default SQL Server providers.

Let's get started!

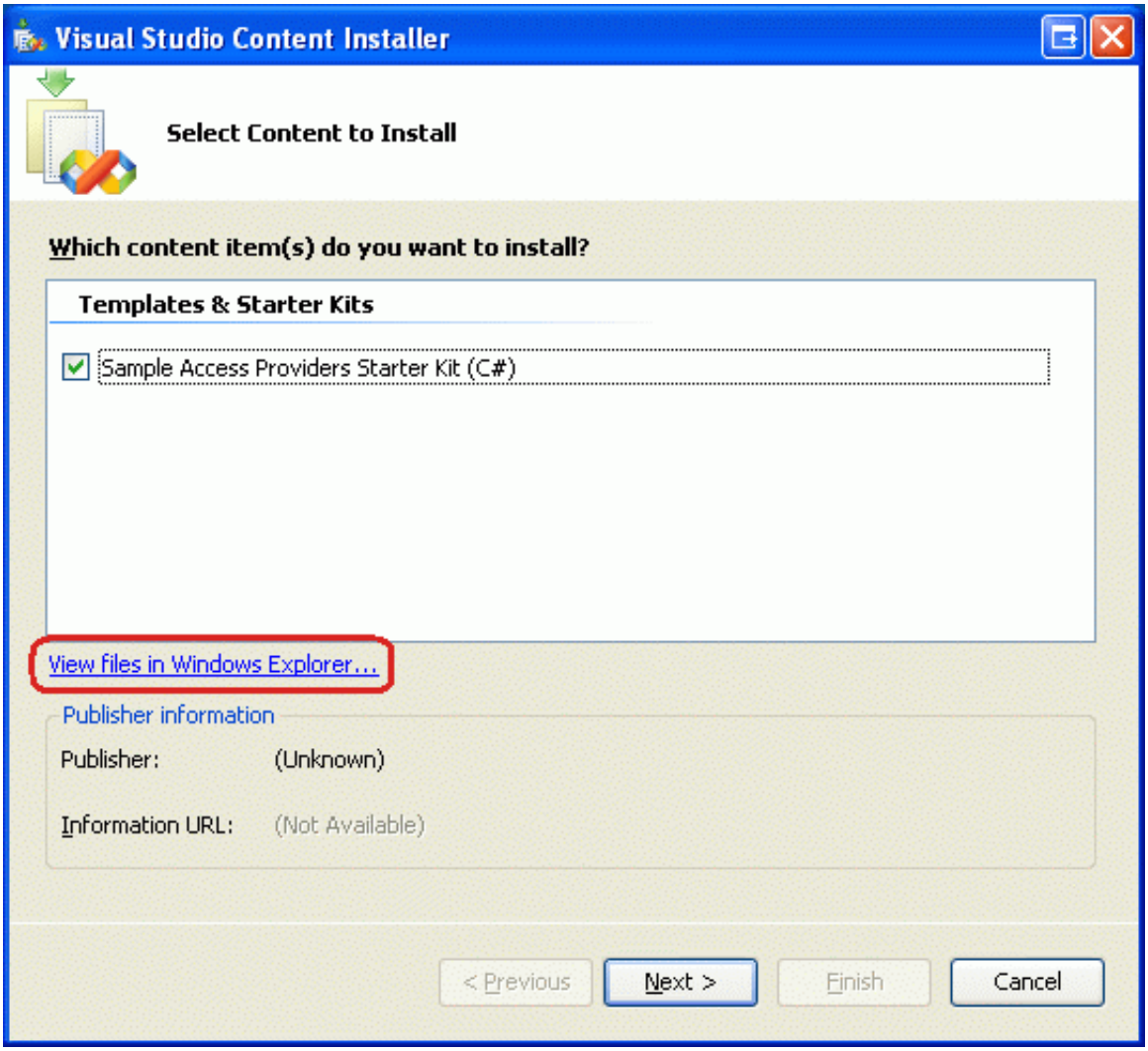
Step 1: Getting the Access Provider Classes into the Web Application

After downloading the VSI file, double-click it to start the installation process. The installation process installs the files to the appropriate folder (`My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual Web Developer\Starter Kits`, on my computer) and is available as a template from the New Projects dialog box as the following screenshot illustrates:



Creating a new project using the ASP.NET Access Providers template creates a Class Library project. The idea here is that you will build this Class Library project, resulting in an assembly, like `AccessProviders.dll`. Once you have this DLL, you can copy it to your web application's `/bin` directory, which will allow you to start using the Access providers for Membership, Roles, and Profile. Compiling the code into an assembly and then using that assembly in the `/bin` folder of those web applications that need to use the Access providers is the ideal approach, as it simplifies deployment and maintainability.

Alternatively, you can copy the C# class files used by the template into your web application's `App_Code` folder. The easiest way to accomplish this is to extract the class files from the VSI file, which you can do from the installation wizard's first screen: choose to "View files in Windows Explorer". The VSI file includes a ZIP file (`ASP.NET Access Providers.zip`) that contains the needed classes in its `Samples` directory. Copy these classes into your web application's `App_Code` folder.



Step 2: Readying the Database

The VSI file includes a Microsoft Access database (`ASPNetDB.mdb`) with the needed tables and views. This database needs to be available to the web application. You can get your hands on this Access database file by viewing the VSI's contents through Windows Explorer, or you can locate the file on your hard drive after installing the VSI package. In either case, once you have located the `ASPNetDB.mdb` copy it to your web application's `App_Data` folder. Make sure that the file is not marked Read Only and that the appropriate user account has read and write permissions on the `App_Data` folder.

Step 3: Updating `Web.config`

The final is to update the `Web.config` file to instruct the application to use the Access-based providers for Membership, Roles, and Profile as opposed to the default SQL Server-based ones. We also need to add a new entry to the `<connectionStrings>` section that references the path of the Access database.

First, add the new `<connectionStrings>` entry. Simply provide the path to the Access database. Assuming you've left the database file name as `ASPNetDB.mdb` and have placed it in the `App_Data` folder, you can use the following markup:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <connectionStrings>
      <add name="LocalAccessDatabase"
connectionString="~/App_Data/ASPNetDB.mdb" providerName="System.Data.OleDb"/>
   </connectionStrings>

   <system.web>
```

```
      ...
   </system.web>
</configuration>
```

Then, in the `<system.web>` element, add the Membership, Roles, and Profile elements, as needed. For example, the following markup clears out the set of Membership providers and then adds the `AccessMembershipProvider`, making it the default Membership provider for the application.

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <connectionStrings>
      <add name="LocalAccessDatabase" connectionString="~/App_Data/ASPNetDB.mdb"
providerName="System.Data.OleDb"/>
   </connectionStrings>

   <system.web>
      <membership defaultProvider="AccessMembershipProvider">
         <providers>
            <clear/>
            <add name="AccessMembershipProvider"
               type="Samples.AccessProviders.AccessMembershipProvider, AssemblyName"
               connectionStringName="LocalAccessDatabase"
               enablePasswordRetrieval="false"
               enablePasswordReset="false"
               requiresUniqueEmail="false"
               requiresQuestionAndAnswer="false"
               minRequiredPasswordLength="1"
               minRequiredNonalphanumericCharacters="0"
               applicationName="SampleSite"
               hashAlgorithmType="SHA1"
               passwordFormat="Hashed"/>
         </providers>
      </membership>
   </system.web>
</configuration>
```

Note that the `name` used in the `<connectionStrings>` section is the same name referenced in the `connectionStringName` attribute of the `<add>` element (`LocalAccessDatabase`). Also, the `type` attribute must be the fully-qualified type for the Membership provider class. If you compiled the classes into an assembly and copied that assembly into the web application's `/bin` directory or installed it in the GAC, then use "`Samples.AccessProviders.AccessMembershipProvider, AssemblyName`" as the `type` value. If you simply copied the class files into the `App_Code` folder, then use just "`Samples.AccessProviders.AccessMembershipProvider`".

Putting It All Together...

Once you've completed these three steps, you can use the Membership, Roles, and Profile system in the same manner as if the default SQL Server-based providers are being used. (This is not entirely true; see the caveats below and in the VSI file's README file.) That means that you can use the Security Web controls - Login, LoginView, CreateUserWizard, and so forth - as well as the Membership, Roles, and Profile APIs. The download at the end of this article includes a sample website that uses the Membership and Roles functionality using the Access providers.

Caveats of Using the Access-based Providers

The README file included in the VSI download notes a few shortcomings of the Access Membership provider, which I repeat here:

Password strength requirements are not enforced in the CreateUser and ChangePassword methods. The settings are read from configuration, but at are not used by the provider.

Account lockouts are not implemented in the provider. The UnlockUser method has no effect, and the provider does not count bad password or bad password answer attempts.

The provider does not support creating a new user with an explicit UserId (i.e. providerUserKey).

The provider does not support retrieving a user based on UserId (i.e. providerUserKey).

See the README file for more detailed information about installation and limitations of these providers.

Conclusion

ASP.NET 2.0's provider model allows for well-defined systems - like Membership, Roles, and Profile - that support a standard API and common Web controls, but enables the concrete implementation to be customized and adjusted at runtime. In this article we examined this inherent flexibility by exploring the Access providers for Membership, Roles, and Profile. These providers, made available by Microsoft, enable developers to use an Access database as the backing store for the Membership, Roles, and Profile systems.

Happy Programming!

- By Scott Mitchell

Attachments

- Download the code used in this article (includes the Sample Access Providers VSI file)

Further Readings

- Forms Authentication, Authorization, Membership, and Roles Tutorials (includes VB & C# versions!)
- ASP.NET 2.0 Membership, Roles, Forms Authentication, and Security Resources
- An Overview of the Provider Model
- Download for Sample Access Providers

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 8
Article Author:	Scott Mitchell
Published Date:	January 3, 2007
Article URL:	http://www.4GuysFromRolla.com/articles/010307-1.aspx